# CIFParse
# A Library of Access Tools for mmCIF
# Reference Guide

**CIFParse Version 6.0a**

**Version 6.0a July 2000**

**Olivera Tosic**
**John D. Westbrook**
Nucleic Acid Database Project
Department of Chemistry and Chemical Biology
Rutgers, The State University of New Jersey

Please direct comments on this document to sw-help@rcsb.rutgers.edu.

**CIFParser - A Library of Access Tools for mmCIF**

Copyright © 1999-2002 Rutgers,The State University of New Jersey

This software is provided WITHOUT WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR ANY OTHER WARRANTY, EXPRESS OR IMPLIED. RUTGERS MAKE NO REPRESENTATION OR WARRANTY THAT THE SOFTWARE WILL NOT INFRINGE ANY PATENT, COPYRIGHT OR OTHER PROPRIETARY RIGHT.

The user of this software shall indemnify, hold harmless and defend Rutgers, its governors, trustees, officers, employees, students, agents and the authors against any and all claims, suits, losses, liabilities, damages, costs, fees, and expenses including reasonable attorneys' fees resulting from or arising out of the use of this software. This indemnification shall include, but is not limited to, any and all claims alleging products liability.

This software may be used only for not-for-profit educational and research purposes.

# Contents

**1 Introduction** **5**

**2 TblFileObj Public Method and Member Descriptions** **6**

2.1 Constructors . . . . . 7

2.1.1 TblFileObj . . . . . 7

2.2 Methods . . . . . 9

2.2.1 OpenFile . . . . . 9

2.2.2 CloseFile . . . . . 10

2.2.3 WriteTable . . . . . 11

2.2.4 IsTablePresent . . . . . 13

2.2.5 GetTablePtr . . . . . 15

2.2.6 GetTable . . . . . 16

2.2.7 DeleteTable . . . . . 17

2.2.8 GetTableNames . . . . . 19

2.2.9 GetBlockNames . . . . . 21

**3 CifFileObjBase Public Method and Member Descriptions** **22**

3.1 Constructors . . . . . 23

3.1.1 CifFileObjBase . . . . . 23

3.2 Methods . . . . . 25

3.2.1 Write . . . . . 25

**4 CifFileObj Public Method and Member Descriptions** **28**

4.1 Constructors . . . . . 29

4.1.1 CifFileObj . . . . . 29

4.2 Methods . . . . . 31

4.2.1 Read . . . . . 31

**5 DDLFileObj Public Method and Member Descriptions** **33**

5.1 Constructors . . . . . 34

5.1.1 DDLFileObj . . . . . 34
5.2 Methods . . . . . 36
5.2.1 Read . . . . . 36
5.2.2 Write . . . . . 38

**6 DICFileObj Public Method and Member Descriptions 40**
6.1 Constructors . . . . . 41
6.1.1 DICFileObj . . . . . 41
6.2 Methods . . . . . 43
6.2.1 Read . . . . . 43
6.2.2 Write . . . . . 45
6.2.3 Compress . . . . . 47

# 1 Introduction

The CIFParser package contains several files that converge together in the TblFileObj class, CifFileObjBase class CifFileObj class, DDLFileObj clas and DICFileObj class. These classes have a number of characteristics that make it useful for a wide variety of applications:

- TblFileObj is developed in order to provide manipulation with multiple datablocks and multiple tables.
- TblFileObj objects provide persistent storage management of themselves.
- CifFileObjBase inherits TblFileObj class. This is base class for all other clases that read mmCif file or ddl/dictionary file.
- CifFileObj inherits CifFileObjBase. This class provides methods to read and write mmCIF data files that conform to the subset of STAR syntax rules adopted by mmCIF.
- DDLFileObj inherits CifFileObjBase. This class provides methods to read and write DDL data files. Each DDL definition is enclosed in a STAR save frame cell.
- DICFileObj inherits CifFileObjBase. This class provides methods to read and write Dictionary data files. Dictionary definition is also enclosed in a STAR save frame cell. Imlicitly defined data are determined from DDL.

This document describes the public interface to all classes.

# 2 TblFileObj Public Method and Member Descriptions

The following section describes the public interface to the TblFileObj class. Example code will be provided along with each method or member.

## 2.1 Constructors

This subsection contains the constructors for the TblFileObj class.

### 2.1.1 TblFileObj

**NAME**

*TblFileObj*

**PROTOTYPE**

```
#include "TblFileObj.h"

TblFileObj::TblFileObj();
TblFileObj::TblFileObj(char * filename, int openMode);
```

**EXAMPLE**

```
#include "TblFileObj.h"

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);
```

**PURPOSE**

*TblFileObj()* is the default constructor for the TblFileObj class.
*TblFileObj(char * , int)* is a constructor for the TblFileObj class that opens a file *filename* in a mode specified by *openMode*. The data will write (WRITE_MODE) in, append (APPEND_MODE) to or read (READ_MODE) from *filename*. *filename* is name of binary file.

**RECEIVES**

TblFileObj()
No Arguments

TblFileObj(char * , int)

| | |
|---|---|
| `filename` | The name of file |
| `openMode` | Mode in which file is opened |

## RETURN VALUE

None

## REMARKS

None

## 2.2 Methods

This subsection contains the public methods for the TblFileObj class.

### 2.2.1 OpenFile

**NAME**

*OpenFile*

**PROTOTYPE**

```
#include "TblFileObj.h"

int TblFileObj::OpenFile(char * filename, int openModeS);
```

**PURPOSE**

*OpenFile(char *, int)* opens a file *filename* in a mode specified by *openMode*. The data will write (WRITE_MODE) in, append (APPEND_MODE) to or read (READ_MODE) from *filename*. *filename* is name of binary file. This method is always called inside constructor, so there is not need to call this method independently.

**RECEIVES**

OpenFile(char *, int)

| | |
|---|---|
| `filename` | The name of file |
| `openMode` | Mode in which file is opened |

**RETURN VALUE**

A negative value indicates an error or warning.

**REMARKS**

See also: *CloseFile*

### 2.2.2 CloseFile

#### NAME

*CloseFile*

#### PROTOTYPE

```
#include "TblFileObj.h"
// this is defined in TblFileObj.h
//  static const int  MAKE_BIN = 1;
//  static const int  DONT_MAKE_BIN = 0;


void TblFileObj::CloseFile(int makeBinFile= MAKE_BIN,int shrink=0);
```

#### EXAMPLE

```
#include "TblFileObj.h"
int num;
char ** names;

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
names =  fobjR->GetBlockNames(num);
for (int i=0; i<num; i++)
   cout<<names[i]<<endl;

fobjR->CloseFile(1);
if (fobjR) delete fobjR;
```

#### PURPOSE

*CloseFile* Writes modified tables into the binary file and closes that file if $makeBinFile = 1$, otherwise binary file is not created.

#### RECEIVES

| | |
|---|---|
| `*makeBinFile` | if this is set to 1, binary file will be created |
| `*shrink` | indicates if file should be shrunk (suggestion-not use this atribute i.e leave it 0) |

**RETURN VALUE**

None

**REMARKS**

None

### 2.2.3 WriteTable

**NAME**

*WriteTable*

**PROTOTYPE**

```
#include "TblFileObj.h"

int TblFileObj::WriteTable(SSTable * s,
                           const char *blockName,
                           const char *tableName,
                           int saveTablePtr=1);
int TblFileObj::WriteTable(SSTable * s,
                           const char *tableName);
```

**EXAMPLE**

```
#include "TblFileObj.h"

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
```

## PURPOSE

*WriteTable (SSTable *, const char *, const char *, int)* puts the table in a list of tables in the TblFileObj object and writes table in the binary file specified by constructor for TblFileObj object. Table will have name *tableName* and will belong to datablock **blockName*.

*WriteTable (SSTable *, const char *)* puts the table in a list of tables in the TblFileObj object and writes table in the binary file specified by constructor for TblFileObj object. Table will have name *tableName* and will belong to current datablock.

## RECEIVES

WriteTable(SSTable *, const char *, const char *, int)

| | |
|---|---|
| `s` | Pointer to SSTable object |
| `*blockName` | Name of datablock |
| `*tableName` | Table name |
| `saveTablePtr` | indicate if the poiter of table is saved for later use. Deafult value is 1. Recommendation: do not set it on 0 |

WriteTable(SSTable *, const char *)

| | |
|---|---|
| `s` | Pointer to SSTable object |
| `*tableName` | Table name |

## RETURN VALUE

A negative value indicates an error or warning.

## REMARKS

None

### 2.2.4 IsTablePresent

#### NAME

*IsTablePresent*

#### PROTOTYPE

```
#include "TblFileObj.h"

int TblFileObj::IsTablePresent(const char *blockName,
                               const char *tableName)
int TblFileObj::IsTablePresent(const char *tableName)
```

#### EXAMPLE

```
#include "TblFileObj.h"

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
if (fobjR->isTablePresent("MyDBlock","MyTable"))
   cout<<"Table exist"<<endl;
```

#### PURPOSE

*IsTablePresent(const char *, const char *)* checks if table *tableName* exists in *blockName* in TblFileObj object. *IsTablePresent(const char *)* checks if table *tableName* exists in current datablock in TblFileObj object.

#### RECEIVES

IsTablePresent(const char *, const char *)

| | |
|---|---|
| `*blockName` | Name of datablock |
| `*tableName` | Table name |

IsTablePresent(const char *)

| | |
|---|---|
| `*tableName` | Table name |

**RETURN VALUE**

Returns 0 when table dosn't exist, 1 when the table exists

**REMARKS**

None

### 2.2.5 GetTablePtr

#### NAME

*GetTablePtr*

#### PROTOTYPE

```
#include "TblFileObj.h"

SSTable *  TblFileObj::GetTablePtr(const char *blockName,
                                   const char *tableName)
```

#### EXAMPLE

```
#include "TblFileObj.h"

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
s = fobjR->GetTablePtr("MyDBlock","MyTable"));
```

#### PURPOSE

*GetTablePtr(const char *, const char *)* gets pointer of table from TblFileObj with specified *blockName* and *tableName*.

#### RECEIVES

| | |
|---|---|
| `*blockName` | Name of datablock |
| `*tableName` | Table name |

#### RETURN VALUE

Returns poiter to SSTable if table exist, otherwise NULL.

#### REMARKS

Table is not copied, just poiter is returned. This poiter must not be deleted because the table will also be deleted form TblFileObj object.

### 2.2.6 GetTable

#### NAME

*GetTablePtr*

#### PROTOTYPE

```
#include "TblFileObj.h"

SSTable *  TblFileObj::GetTable(const char *blockName,
                                const char *tableName)
```

#### EXAMPLE

```
#include "TblFileObj.h"

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
s = fobjR->GetTable("MyDBlock","MyTable"));
```

#### PURPOSE

*GetTable(const char *,const char )* gets pointer of table from TblFileObj with specified *blockName* and *tableName*, but actually copies the table.

#### RECEIVES

| | |
|---|---|
| `*blockName` | Name of datablock |
| `*tableName` | Table name |

#### RETURN VALUE

Returns poiter to SSTable.

#### REMARKS

This method copies the table and this pointer should be deleted.

### 2.2.7 DeleteTable

#### NAME

*DeleteTable*

#### PROTOTYPE

```
#include "TblFileObj.h"

int TblFileObj::DeleteTable(const char *blockName,
                            const char *tableName)
int TblFileObj::DeleteTable(const char *tableName)
```

#### EXAMPLE

```
#include "TblFileObj.h"

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
fobjR->DeleteTable("MyDBlock","MyTable"));
```

#### PURPOSE

*DeleteTable(const char *, const char *)* deletes table with specified *blockName* and *tableName*, form TblFileObj object. *DeleteTable(const char *)* deletes table from current datablock with specified *tableName*.

#### RECEIVES

DeleteTable(const char *, const char *)

| | |
|---|---|
| `*blockName` | Name of datablock |
| `*tableName` | Table name |

DeleteTable(const char *)

| | |
|---|---|
| `*tableName` | Table name |

## RETURN VALUE

A negative value indicates an error or warning.

## REMARKS

None

### 2.2.8 GetTableNames

#### NAME

*GetTableNames*

#### PROTOTYPE

```
#include "TblFileObj.h"

char **  TblFileObj::GetTableNames(const char *blockName,
                                   int num);
```

#### EXAMPLE

```
#include "TblFileObj.h"
int num;
char ** names;

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable1");
fobjR->WriteTable(s,"MyDBlock","MyTable2");
names = fobjR->GetTableNames("MyDBlock", num);
for (int i=0; i<num; i++)
   cout<<names[i]<<endl;
```

#### PURPOSE

*GetTableNames* Gets all table names for specified *blockName*.

#### RECEIVES

| | |
|---|---|
| `*blockName` | Name of datablock |

#### RETURN VALUE

Returns pointer to list of strings which represents a list of table names and number of tables.

**REMARKS**

See also: *GetBlockNames*

### 2.2.9 GetBlockNames

#### NAME

*GetBlockNames*

#### PROTOTYPE

```
#include "TblFileObj.h"

char **  TblFileObj::GetBlockNames(int num);
```

#### EXAMPLE

```
#include "TblFileObj.h"
int num;
char ** names;

TblFileObj * fobjR = new TblFileObj("./test/TESTFILE1", WRITE_MODE);

s = new SSTable("MyTable");
FillTestTable(s, nRows, nCols, "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345689");
fobjR->WriteTable(s,"MyDBlock","MyTable");
names =  fobjR->GetBlockNames(num);
for (int i=0; i<num; i++)
   cout<<names[i]<<endl;
```

#### PURPOSE

*GetTableNames* Gets all blocks names.

#### RECEIVES

| | |
|---|---|
| `*blockName` | Name of datablock |

#### RETURN VALUE

Returns pointer to list of strings which represents a list of block names and number of blocks.

#### REMARKS

None

# 3 CifFileObjBase Public Method and Member Descriptions

The following section describes the public interface to the CifFileObjBase class. Example code will be provided along with each method or member. CifFileObjBase class inherits TblFileObj class and provides reading mmCIF files into TblFileObj object.

## 3.1 Constructors

This subsection contains the constructors for the CifFileObjBase class.

### 3.1.1 CifFileObjBase

**NAME**

*CifFileObjBase*

**PROTOTYPE**

```
#include "CifFileObjBase.h"

CifFileObjBase::CifFileObjBase();
CifFileObjBase::CifFileObjBase(char * objFileName, int openMode,
                               int maxLineLength = 80,
                               const char * nulChar = "?", int verbose = 0);
```

**EXAMPLE**

```
#include "CifFileObjBase.h"

CifFileObjBase * fobjR = new CifFileObjBase("./test/TESTFILE1", WRITE_MODE);
```

**PURPOSE**

*CifFileObjBase()* is the default constructor for the CifFileObjBase class.

*CifFileObjBase(char * , int, int,const char *, int)* is a constructor for the class that opens a file *filename* in a mode specified by *openMode*.

**RECEIVES**

CifFileObjBase()

No Arguments

CifFileObjBase(char * , int, int,const char *, int)

| | |
|---|---|
| `objfilename` | The name of binary file |
| `openMode` | Mode in which file is opened |
| `maxLineLength` | Maximal length of a line in a cif file (used when we want to write data in a cif file) |
| `nulChar` | If there is no value for an item, then nullChar will be put as a value |
| `verbose` | a non-zero value activates verbose output of diagnostic and informational messages from all library functions |

**RETURN VALUE**

None

**REMARKS**

WRITE_MODE - in this mode read method reads mmCIF file and *writes* data into binary file.

READ_MODE - in this mode read method *reads* binary file.

APPEND_MODE - in this mode read method reads binary file and data can be *appended* to the file.

## 3.2 Methods

This subsection contains the public methods for the CifFileObjBase class.

### 3.2.1 Write

#### NAME

*Write*

#### PROTOTYPE

```
#include "CifFileObjBase.h"
char *diags=NULL;

int CifFileObjBase::Write(const char * cifFileName,
                          int sortFlag =0, int emptyTable =0);
int CifFileObjBase::Write(const char * cifFileName,
                          ReVarCifArray<CifString> & catOrder,
                          int emptyTable =0);
```

#### EXAMPLE

```
#include "CifFileObjBase.h"
char * diags;

CifFileObjBase * fobjR = new CifFileObjBase("./test/TESTFILE1", WRITE_MODE);
   .
   .
   .
fobj->Write("./test/Test1.ocif");

#include "CifFileObjBase.h"
ReVarCifArray<CifString> catOrder;

fobjR = new CifFileObjBase("./test/Test.db", WRITE_MODE, 80, "?", 0);
   .
   .
   .
catOrder.Add("struct");
catOrder.Add("diffrn");
catOrder.Add("entity");

fobjR->Write("./test/Test.ocif",catOrder);
```

## PURPOSE

*Write(char *, char *&, int)* writes CifFileObjBase object into a cif file, specified by *cifFileName*.

*Write(char *, ReVarCifArray<CifString> &, int)* writes CifFileObjBase object into a cif file, specified by *cifFileName* in a specified order *catOrder*.

## RECEIVES

Write(char * , char *&, int)

| | |
|---|---|
| `cifFileName` | The name of output cif file |
| `sortFlag` | If it zero, datablocks and table will be written in a original order, as they are created. If it 1 then datablocks, and tables in datablock will be in alfabetical order. |
| `emptyTable` | If it zero, empty tables will not be written in a ciff file (tables with no rows). If it 1 then empty table will be writen as a table with one row, and all values will be *nullChar* (See Constuctor). |

Write(char * , ReVarCifArray<CifString> &, int)

| | |
|---|---|
| `cifFileName` | The name of output cif file |
| `catOrder` | List of cattegory(table) names which we want to write at the begining of cif file. Other tables will be written in original order. |
| `emptyTable` | If it zero, empty tables will not be written in a ciff file (tables with no rows). If it 1 then empty table will be writen as a table with one row, and all values will be *nullChar* (See Constuctor). |

## RETURN VALUE

Returns 1 upon success, otherwise 0.

**REMARKS**

None

# 4 CifFileObj Public Method and Member Descriptions

The following section describes the public interface to the CifFileObj class. Example code will be provided along with each method or member. CifFileObj class inherits CifFileObjBase class and provides reading mmCIF files into TblFileObj object.

## 4.1 Constructors

This subsection contains the constructors for the CifFileObj class.

### 4.1.1 CifFileObj

#### NAME

*CifFileObj*

#### PROTOTYPE

```
#include "CifFileObj.h"

CifFileObj::CifFileObj();
CifFileObj::CifFileObj(char * objFileName, int openMode,
                       int maxLineLength = 80,
                       const char * nulChar = "?", int verbose = 0);
```

#### EXAMPLE

```
#include "CifFileObj.h"

CifFileObj * fobjR = new CifFileObj("./test/TESTFILE1", WRITE_MODE);
```

#### PURPOSE

*CifFileObj()* is the default constructor for the CifFileObj class.
*CifFileObj(char * , int, int,const char *, int)* is a constructor for the class that opens a file *filename* in a mode specified by *openMode*.

#### RECEIVES

CifFileObj()
No Arguments

CifFileObj(char * , int, int,const char *, int)

| | |
|---|---|
| `objfilename` | The name of binary file |
| `openMode` | Mode in which file is opened |
| `maxLineLength` | Maximal length of a line in a cif file (used when we want to write data in a cif file) |
| `nulChar` | If there is no value for an item, then nullChar will be put as a value |
| `verbose` | a non-zero value activates verbose output of diagnostic and informational messages from all library functions |

**RETURN VALUE**

None

**REMARKS**

WRITE_MODE - in this mode read method reads mmCIF file and *writes* data into binary file.

READ_MODE - in this mode read method *reads* binary file.

APPEND_MODE - in this mode read method reads binary file and data can be *appended* to the file.

## 4.2 Methods

This subsection contains the public methods for the CifFileObj class.

### 4.2.1 Read

**NAME**

*Read*

**PROTOTYPE**

```
#include "CifFileObj.h"

int CifFileObj::Read(char * cifFileName, char *& diagnostics);
int CifFileObj::Read(char * cifFileName, char *& diagnostics,
                     int &err, int &warn);
```

**EXAMPLE**

```
#include "CifFileObj.h"
char *diags=NULL;
int err, warn;

CifFileObj * fobjR = new CifFileObj("./test/TESTFILE1", WRITE_MODE);
fobj->Read("./test/Test1.cif",diags, err, warn);
cout<<"diags = "<<diags<<endl;
```

**PURPOSE**

*Read(char *, char *&)* reads cif file, specified by *cifFileName*. *Read(char *, char *&, int, int)* reads cif file, specified by *cifFileName*, and returns number of errors *err* and number of warnings *warn* found in the cif file.

**RECEIVES**

CifFileObj()

No Arguments

CifFileObj(char * , char *&)

| | |
|---|---|
| `cifFileName` | The name of cif file |
| `diagnostics` | Pointer to string |

CifFileObj(char * , char *&, int, int)

| | |
|---|---|
| `cifFileName` | The name of cif file |
| `diagnostics` | Pointer to string |
| `err` | Number of errors |
| `war` | Number of warnings |

**RETURN VALUE**

Returns -1 if the cif file could not be opened, otherwise returns 0. *diagnostics* shows if the method could successfully parse the input file. In that case diagnostics is empty string, otherwise diagnostics contains description of error.

**REMARKS**

None

# 5 DDLFileObj Public Method and Member Descriptions

The following section describes the public interface to the DDLFileObj class. Example code will be provided along with each method or member. DDLFileObj class inherits CifFileObjBase class and provides reading DDL files into TblFileObj object.

## 5.1 Constructors

This subsection contains the constructors for the DDLFileObj class.

### 5.1.1 DDLFileObj

**NAME**

*DDLFileObj*

**PROTOTYPE**

```
#include "DDLFileObj.h"

DDLFileObj::DDLFileObj();
DDLFileObj::DDLFileObj(char * objFileName, int openMode,
                       int maxLineLength = 80,
                       const char * nulChar = "?", int verbose = 0);
```

**EXAMPLE**

```
#include "DDLFileObj.h"

DDLFileObj * fobjR = new DDLFileObj("./test/TESTFILE1", WRITE_MODE);
```

**PURPOSE**

*DDLFileObj()* is the default constructor for the DDLFileObj class. *DDLFileObj(char * , int, int,const char *, int)* is a constructor for the class that opens a file *filename* in a mode specified by *openMode*.

**RECEIVES**

DDLFileObj()

No Arguments

DDLFileObj(char * , int, int,const char *, int)

| | |
|---|---|
| `objfilename` | The name of binary file |
| `openMode` | Mode in which file is opened |
| `maxLineLength` | Maximal length of a line in a cif file (used when we want to write data in a cif file) |
| `nulChar` | If there is no value for an item, then nullChar will be put as a value |
| `verbose` | a non-zero value activates verbose output of diagnostic and informational messages from all library functions |

**RETURN VALUE**

None

**REMARKS**

WRITE_MODE - in this mode read method reads mmCIF file and *writes* data into binary file.

READ_MODE - in this mode read method *reads* binary file.

APPEND_MODE - in this mode read method reads binary file and data can be *appended* to the file.

## 5.2 Methods

This subsection contains the public methods for the DDLFileObj class.

### 5.2.1 Read

#### NAME

*Read*

#### PROTOTYPE

```
#include "DDLFileObj.h"

int DDLFileObj::Read(char * ddlFileName,
                     ISTable *&format,
                     char *& diagnostics);
```

#### EXAMPLE

```
#include "DDLFileObj.h"
char *diags=NULL;
DDLFileObj * fobjR = NULL;
ISTable *format;

format = new ISTable("format");
fobjR = new DDLFileObj("./test/TESTFILE6", WRITE_MODE, 80, "?", 1);
fobjR->Read("./test/ndb_ddl",format, diags);
cout<<"diags = "<<diags<<endl;
```

#### PURPOSE

*Read(char *, ISTable *&, char *&)* reads ddl file, specified by *ddlFileName*. Also, returns information about format of save frames in table *format* i.e. what information are relevant for category save frames what for item save frames. This informatin is needed when data have to be written back in ddl file in a same format.

#### RECEIVES

Read(char * , ISTable *&, char *&)

| | |
|---|---|
| `ddlFileName` | The name of ddl file |
| `format` | Pointer ISTable that keep information of save frame format |
| `diagnostics` | Pointer to string |

**RETURN VALUE**

Returns -1 if the cif file could not be opened, otherwise returns 0. *diagnostics* shows if the method could successfully parse the input file. In that case diagnostics is empty string, otherwise diagnostics contains description of error.

**REMARKS**

None

### 5.2.2 Write

#### NAME

*Write*

#### PROTOTYPE

```
#include "DDLFileObj.h"

int DDLFileObj::Write(const char * ddlFileName,ISTable * format);
```

#### EXAMPLE

```
#include "DDLFileObj.h"
char * diags;
ISTable *format;

format = new ISTable("format");
DDLFileObj * fobjR = new DDLFileObj("./test/TESTFILE1", WRITE_MODE);
fobj->Read("./test/Test1.cif",format,diags);
cout<<"diags = "<<diags<<endl;
fobj->Write("./test/Test1.ocif",format);
```

#### PURPOSE

*Write(char *, ISTable *)* writes DDLFileObj object into a ddl file, i.e. save frame formated file, specified by *ddlFileName*. Format is read from table *format*.

#### RECEIVES

Write(char * , char *&, int)

| | |
|---|---|
| `ddlFileName` | The name of output DDL file |
| `format` | Pointer ISTable that keep information of save frame format |

#### RETURN VALUE

Returns 1 upon success, otherwise 0.

**REMARKS**

None

# 6 DICFileObj Public Method and Member Descriptions

The following section describes the public interface to the DICFileObj class. Example code will be provided along with each method or member. DICFileObj class inherits CifFileObjBase class and provides reading dictionary files into TblFileObj object.

## 6.1 Constructors

This subsection contains the constructors for the DICFileObj class.

### 6.1.1 DICFileObj

**NAME**

*DICFileObj*

**PROTOTYPE**

```
#include "DICFileObj.h"

DICFileObj::DICFileObj();
DICFileObj::DICFileObj(char * objFileName, int openMode,
                       int maxLineLength = 80,
                       const char * nulChar = "?", int verbose = 0);
```

**EXAMPLE**

```
#include "DICFileObj.h"

DICFileObj * fobjR = new DICFileObj("./test/TESTFILE1", WRITE_MODE);
```

**PURPOSE**

*DICFileObj()* is the default constructor for the DICFileObj class. *DICFileObj(char * , int, int,const char *, int)* is a constructor for the class that opens a file *filename* in a mode specified by *openMode*.

**RECEIVES**

DICFileObj()

No Arguments

DICFileObj(char * , int, int,const char *, int)

| | |
|---|---|
| `objfilename` | The name of binary file |
| `openMode` | Mode in which file is opened |
| `maxLineLength` | Maximal length of a line in a cif file (used when we want to write data in a cif file) |
| `nulChar` | If there is no value for an item, then nullChar will be put as a value |
| `verbose` | a non-zero value activates verbose output of diagnostic and informational messages from all library functions |

**RETURN VALUE**

None

**REMARKS**

WRITE_MODE - in this mode read method reads mmCIF file and *writes* data into binary file.

READ_MODE - in this mode read method *reads* binary file.

APPEND_MODE - in this mode read method reads binary file and data can be *appended* to the file.

## 6.2 Methods

This subsection contains the public methods for the DICFileObj class.

### 6.2.1 Read

#### NAME

*Read*

#### PROTOTYPE

```
#include "DICFileObj.h"

int DICFileObj::Read(char * dicFileName, TblFileObj * ddl,
                     ISTable *&format,char *& diagnostics);
```

#### EXAMPLE

```
#include "DICFileObj.h"
#include "DDLFileObj.h"
char *diags=NULL;
ISTable *format;
ISTable *ddlformat;
DDLFileObj * ddl = NULL;

ddl = new DDLFileObj("ddl", WRITE_MODE, 80, "?", 1);
ddlformat = new ISTable("ddlformat");
ddl->Read("./test/ddl-mm",ddlformat, diags);
cout<<"diags = "<<diags<<endl;

format = new ISTable("format");
DICFileObj * fobjR = new DICFileObj("./test/TESTFILE1", WRITE_MODE);
fobj->Read("./test/Test1.cif",ddl , format,diags);
cout<<"diags = "<<diags<<endl;
```

#### PURPOSE

*Read(char *, TblFileObj *, ISTable *&, char *&)* reads dictionary, specified by *dicFileName*. Implicit values determines from *ddl*. Return format of save frames in *format*.

**RECEIVES**

Read(char *, TblFileObj *, ISTable *&, char *&)

| | |
|---|---|
| `dicFileName` | The name of dictionary file |
| `ddl` | Pointer to TblFileObj that keeps information from ddl |
| `format` | Pointer ISTable that keep information of save frame format |
| `diagnostics` | Pointer to string |

**RETURN VALUE**

Returns -1 if the cif file could not be opened, otherwise returns 0. *diagnostics* shows if the method could successfully parse the input file. In that case diagnostics is empty string, otherwise diagnostics contains description of error.

**REMARKS**

None

### 6.2.2 Write

#### NAME

*Write*

#### PROTOTYPE

```
#include "DICFileObj.h"

int DICFileObj::Write(const char * dicFileName,
                      TblFileObj * ddl,
                      ISTable * format);
```

#### EXAMPLE

```
char *diags=NULL;
DICFileObj * fobjR = NULL;
ISTable *format;
ISTable *ddlformat;
DDLFileObj * ddl = NULL;

ddl = new DDLFileObj("ddl", WRITE_MODE, 80, "?", 1);
ddlformat = new ISTable("ddlformat");
ddl->Read("./test/ddl-mm",ddlformat, diags);

format = new ISTable("format");
fobjR = new DICFileObj("./test/TESTFILE6", WRITE_MODE, 80, "?", 0);
fobjR->Read("./test/dictionary.dic",ddl , format, diags);

fobjR->Write("./test/view2.dic",ddl,format);
```

#### PURPOSE

*Write(char *, TblFileObj *, ISTable *)* writes DICFileObj object into a ddl file, i.e. save frame formated file, specified by it dicFileName. Format is read from table *format*.

#### RECEIVES

Write(char *, TblFileObj *, ISTable *)

| | |
|---|---|
| `dicFileName` | The name of output dictionary file |
| `ddl` | Pointer to TblFileObj that keeps information from ddl |
| `format` | Pointer ISTable that keep information of save frame format |

**RETURN VALUE**

Returns 1 upon success, otherwise 0.

**REMARKS**

None

### 6.2.3 Compress

#### NAME

*Compress*

#### PROTOTYPE

```
#include "DICFileObj.h"

void DICFileObj::Compress(TblFileObj * ddl);
```

#### EXAMPLE

```
char *diags=NULL;
DICFileObj * fobjR = NULL;
ISTable *format;
ISTable *ddlformat;
DDLFileObj * ddl = NULL;

ddl = new DDLFileObj("ddl", WRITE_MODE, 80, "?", 1);
ddlformat = new ISTable("ddlformat");
ddl->Read("./test/ddl-mm",ddlformat, diags);

format = new ISTable("format");
fobjR = new DICFileObj("./test/TESTFILE6", WRITE_MODE, 80, "?", 0);
fobjR->Read("./test/dictionary.dic",ddl , format, diags);

fobjR->Compress(ddl);

fobjR->Write("./test/view2.dic",ddl,format);
fobjR->CloseFile(1);
if (fobjR) delete fobjR;
```

#### PURPOSE

*Compress(char *, TblFileObj *, ISTable *)* compresses dictionary i.e removes redundant rows from tables. Redundan rows are rows that have same key. What is key for paricular table is found in *ddl*.

#### RECEIVES

Write(char *, TblFileObj *, ISTable *)

| | |
|---|---|
| `ddl` | Pointer to TblFileObj that keeps information from ddl |

**RETURN VALUE**

Returns 1 upon success, otherwise 0.

**REMARKS**

None